Defense Information Infrastructure (DII)

Common Operating Environment (COE)

Version 3.0/B

Programming Guide (Digital UNIX and AIX)

February 19, 1997

Prepared for:

Defense Information Systems Agency

Prepared by:

Inter-National Research Institute (INRI)
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 20191

Table of Contents

List of Tables

List of Figures

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
`Courier Font`	Used to indicate entries to be typed at the keyboard, UNIX commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: `tar xvf /dev/rmt/3mn`
"Quotation Marks"	Used to indicate prompts and messages that appear on the screen.
Italics	Used for emphasis.

This page intentionally left blank.

1. Writing Programs Using the COE Tools

1.1 Overview

This document provides an introduction to the capabilities of the Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0/B tools for the Digital UNIX 4.0 Operating System and the AIX 4.1.4 Operating System. These tools consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. It explains the basic use of the tools, regardless of whether they are run from a menu or from the command line. The document consists of the following sections and appendices:

Descriptions assume familiarity with the C programming language and with the UNIX development environment.

1.2 Additional Sources of Information

Reference the following documents for more information about the DII COE toolkit:

- *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0/B Application Programmer Interface (API) Reference Guide (Digital UNIX and AIX)*, DII.30B.DECIBM.RG-1, Inter-National Research Institute, February 19, 1997

- *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification* Version 2.0, DII COE I&RTS:Rev 2.0, Inter-National Research Institute, October 23, 1995.

2. Application Development Overview

Developers may require access to public APIs to ensure an application complies with the *DII COE Integration and Runtime Specification*. To use the public APIs, developers must compile and link the application with the libraries and header files provided in the Developer's Toolkit. Remember that your `DII_DEV` directory and Motif `include` files and libraries may reside in a different location than in the example compile statements. Public APIs are documented in the *DII COE API Reference Guide (Digital UNIX and AIX).*

2.1 Writing Your Application with the DII COE APIs

To access the DII COE tools through the provided APIs, you must include the following header in your application:

```
#include <DIITools.h>
```

The standard location for the Developer's Toolkit header is:

```
DII_DEV/include
```

The following is an example of using the DII COE COEAskUser tool, which is used to display a question and two possible responses to the user. After the user chooses a response, the response is returned.

```
#include <stdio.h>
#include <DIITools.h>

/*****************************/
/* COEAskUser_example        */
/*****************************/
int main(int argc, char *argv[])
{
char    b1_lab[] = "MY_YES";
char    b2_lab[] = "MY_NO";
char    message[]="This is my test Message";
int     ret_val;

/* Call DII/COE Library Function */
ret_val = COEAskUser, message, b1_lab, b2_lab);

exit(ret_val);
}
```

2.2 Building Your Application with the DII COE APIs

To build your application with DII COE APIs, you must link your application with the `libCOETools.a`, `libCOE.a` , and `libPrintClient` libraries, which are on the DII COE Developer's Toolkit tape.

The standard location for the Developer's Toolkit libraries is:

```
DII_DEV/libs
```

The actual compile and link statement for an application that uses DII COE APIs should resemble the following (substitute your location for the `DII_DEV` directory and Motif libraries and `include` files):

```
cc -Aa -o COEAskUser_example COEAskUser_example.c -I/h/DII_DEV/include
-I/usr/include/Motif1.2 -I/usr/include/X11R5 -L/h/DII_DEV/libs
-lCOETools -lCOE -L/usr/lib/X11R5 -L/usr/lib/Motif1.2 -lXm -lXt -lX11
```

where `COEAskUser_example` is the name of the program being compiled.

2.3 Running Your Application

The DII COE provides the foundation and infrastructure in which one or more applications run. To operate under the COE, applications must be formatted properly as segments. The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are managed most conveniently as a unit. Segments generally are defined to keep related CSCIs together so functionality easily may be included or excluded. All applications must be put in the DII COE runtime environment segment format to be installed onto a DII COE-compliant machine.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, `SegDescrip`, which is called the segment descriptor subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

Once installed, your application can be invoked in the DII COE environment in two ways: (1) running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 4.4, *Customizing Your Segment*, describes how to set up your application to be invoked as a menu item or as an icon.

3. Printer Overview

3.1 COE Print Service

The DII COE Printer API provides a simple, platform-independent method for DII COE applications to print text and graphics data. The API currently consists of 12 C language functions and 3 executable programs.

The COE print service is based on a client-server architecture. Each printer is managed by a single workstation that acts as the server for all print requests for that printer. The print server handles access controls, queue management, and error notification.

Every COE workstation runs a "printer agent," which facilitates communication between client applications and the print server. All printer API functions and executable programs use this printer agent.

3.2 Configuring a New Printer for the DII COE API

The DII COE Printer API consists of the following 12 C language functions:

Low-level functions

```
C  int close_printer(char **file_name, FILE **file_pointer);

C  int get_printer_descriptions(char **c_printer_description);

C  int get_printer_name(char **c_printer_name);

C  int get_printer_type(char **c_printer_type);

C  int open_printer(char *xcp_security_level,
   int xi_line_length,
   int xi_page_length,
   int xi_line_spacing,
   int xi_indent,
   char **xcp_file_name,
   FILE **xfp);

C  int page_break(FILE **xfp);

C  int write_printer(char **c_string, FILE *fp);

C  int write_printer_array(char **c_string, FILE *fp);
```

High-level functions

```
C  VDirectPrintFile(char *filename, int prt_rec)

C  VDirectPrintMsg(**msg_array, int nlines)

C  VPrintFile(char *filename)

C  VPrintMsg(char **msg_array, int nlines)
```

NOTE: Low-level and high-level functions should not be used within the same application.

The `close_printer` function is used to conclude a print job and send the data to the printer. If `close_printer` is not called, the print job will not print.

The `get_printer_descriptions`, `get_printer_name`, and `get_printer_type` functions allow an application to retrieve the name, the type, and a description of the current default printer. All three functions return a string value via the pointer that was passed as an argument to the function. The printer name and description are simple text fields. The printer type is "ASCII", "HPCL", or "PostScript".

The `open_printer` function is used to send text data to a printer. It establishes a print context, including the security level, line length, page length, line spacing, and indentation for the print job. It returns a file pointer through its last argument. This file pointer is used for all subsequent actions on this print job.

The `page_break` function is used to indicate that the lines of text that follow should begin at the top of the next page.

The `write_printer` and `write_printer_array` functions are used to send the actual text data to a previously opened printer context.

The `VDirectPrintFile` and `VPrintFile` functions are used to print text data from a file on disk. The `VDirectPrintMsg` and `VPrintMsg` functions are used to print text data from an array of strings in memory. All four functions generate a security banner at the top and bottom of each output page. On completion, these functions return the internal number of the selected printer or they return `-1` if the user canceled the job or if an error occurred.

The `VPrintFile` and `VPrintMsg` functions provide the user with a `Print Chooser` window, which allows the user to select the destination printer. The `VDirectPrintMsg` function bypasses the `Print Chooser` window and prints directly to the COE default printer. The `VDirectPrintFile` function bypasses the `Print Chooser` window and prints directly to the specified printer. The printer number is specified as a return value from a previous `VPrintXXX` or `VDirectPrintXXX` function call or as `-1` for the COE default printer.

The DII COE Printer API also consists of the following three executable programs:

- EM_get_current_printer_name
- EM_get_current_printer_type
- EM_get_current_printer_desc

The executable programs provide the same functionality as the C functions of the same name. Sample printer programs are shown below.

Sample Printer Programs

```
#include <stdio.h>

#include <Printer/PrintAPI.h>

/* Number of lines in text message */
#define TEXT_LINES  6

static char *PrintMessage[TEXT_LINES] =
{
  "Test message, line 1",
  "Test message, line 2",
  "Test message, line 3",
  "Test message, line 4",
  "Test message, line 5",
  NULL
};

int main(int argc, char *argv[])
{
  if (VPrintMsg(PrintMessage, TEXT_LINES) ==  -1)
    {
      fprintf(stderr, "Printer error.\n");
    }
}
--------------------------------------------------------------------------------
#include <stdio.h>

#include <Printer/PrintAPI.h>

int main(int argc, char *argv[])
{
  int printer_num;

  /* Print a local text file using the system default printer */
  printer_num = VDirectPrintFile("textfile1",  -1);
  if (printer_num == -1)

    {
      fprintf(stderr, "Printer error.\n");
      exit(1);
    }

  /* Print the system hosts file using a user-selected printer */
  printer_num = VPrintFile("/etc/hosts");
  if (printer_num == -1)
    {
      fprintf(stderr, "Printer error.\n");
      exit(2);
    }

  if (printer_num != -1)
    {
      /* Print another text file to the printer that the user just
         selected */
      printer_num = VDirectPrintFile("textfile2", printer_num);
      if (printer_num == -1)
        {
          fprintf(stderr, "Printer error.\n");
          exit(3);
        }
    }
}
```

This page intentionally left blank.

4. Segment Development

The following section discusses the different types of segments and the process of segment creation. Refer to Section 5.0, *Runtime Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of segments.

4.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS (Commercial Off-the-Shelf), Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. For example, a `Scripts` subdirectory is required for an Account Group segment. The `Scripts` subdirectory normally contains scripts such as `.cshrc`, `.xsession`, and `.login`. These scripts serve as a template for establishing a basic runtime environment. For software segments, the `Scripts` subdirectory contains environmental extension files. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.

Figure 1. Segment Directory Structure

The following runtime subdirectories normally are required, depending on the segment type: (1) `SegDescrip`, which is the directory containing segment descriptor files; (2) `Scripts`, which is the directory containing script files; (3) `bin`, which is the directory containing executable programs for the segment; and (4) `data`, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

The `SegDescrip` directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time, while APIs for the segments that own the

data are used at runtime. Refer to Section 5.5, *Segment Descriptors*, of the *DII COE Integration and Runtime Specification* for a detailed explanation of `SegDescrip` files.

4.2 COE Tools Overview

The COE tools were constructed to aid developers in the creation and ultimate installation of DII COE segments. All tools can be run from the command line, and some can be run from other code using published APIs.

4.2.1 Running the COE Tools From the Command Line

This section provides a brief overview of running the COE tools from the command line. Reference the following sources for detailed information about the COE tools: Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*; the Developer's Toolkit release notes; and the help page provided by the tools.

When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools are used to communicate with the outside world in two ways. First, the tools use the exit function to set the UNIX `status` environment variable. The `status` return value is set to `0` for normal tool completion or to `-1` if an error occurs. A `status` return value greater than `0` indicates a completion code that is tool specific.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs.

> **NOTE**: Redirecting `stdin` is not always convenient. The `-R` command line parameter allows a tool to read input from a response file instead of from `stdin`.

For example, the COEPrompt tool displays a message and allows the user to type a response. The user's response, then, is written to `stdout`. The following statement shows how this tool can be used to ask the user to enter the name of a file:

```
COEPrompt "Enter Filename" | MyProg
```

Or, the following statement can be used to write the results to a file:

```
COEPrompt "Enter Filename" > /tmp/tempfile
```

4.2.2 COE Runtime Tools

Reference Appendix C of the *DII COE Integration and Runtime Specification* for a complete description of DII COE runtime tools.

4.2.3 COE Developer's Tools

The MakeAttribs, TestInstall, and TestRemove tools must be run as the `root` user because they modify files the user may not own. ConvertSeg, TimeStamp, VerifySeg, and VerUpdate should also be run as the `root` user, although it is not mandatory. These four tools require the user to have write permission to the segment against which the tool was executed.

Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for a complete description of the DII COE developer's tools.

4.3 Building Your Segment

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, `SegDescrip`, which is the segment descriptor subdirectory.

This section describes a process to turn an application into a segment so it can be a part of the DII COE. As described earlier, a segment is a collection of one or more CSCIs most conveniently managed as a unit.

4.3.1 Identifying and Creating Required Subdirectories

There are six segment types: Account Group, COTS, Data, Database, Software, and Patch. Each segment type is assigned its own subdirectory. Precise files depend on the segment type.

The following subdirectories normally are required:

Subdirectory	Description
`SegDescrip`	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to modify any files directly for resources it does not *own*; in other words, a segment cannot modify files or resources outside an assigned directory. The DII COE tools coordinate the modification of all community files at installation time, while APIs for the segment that owns the data are used at runtime. This subdirectory contains the installation instructions for the segment.
`Scripts`	Subdirectory containing script files. This subdirectory will normally contain scripts such as `.cshrc`, `.xsession`, and `.login`. These scripts serve as a template for establishing a runtime environment.
`bin`	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of segment.
`data`	Subdirectory for static data items, such as menu items, that are

	unique to the segment but that will be the same for all users on all workstations.

Reference Sections 5.0-5.5 of the *DII COE Integration and Runtime Specification* for a detailed explanation of segment directory layout and a description of each `SegDescrip` file.

4.3.2 Creating or Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Reference Table 1 to determine the descriptor files required for each segment type. For example, the AcctGrp segment requires `ReleaseNotes`, `SegInfo`, `SegName`, and `VERSION` descriptor files in the `SegDescrip` directory. Some segment descriptor information is provided within the files listed in Table 1.

NOTE: In Table 1, `Aggregate` and `COE Comp` are segment attributes that can be associated with any type of segment.

File	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W	Patch
DEINSTALL	O	O	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O	O	O
Installed	I	I	I	I	I	I	I	I
PostInstall	O	O	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R	R	R
SegName	R	R	R	R	R	R	R	R
Validated	I	I	I	I	I	I	I	I
VERSION	R	R	R	R	R	R	R	R
R - Required	O - Optional		I - Created by Integrator or Installation Software					

Table 1. Segment Descriptor Files

Other segment descriptor information is arranged within subsections of the `SegInfo` file. As with the descriptor files themselves, some sections of the `SegInfo` file are required and others are optional depending on the type of segment. Table 2 defines the required and optional sections for each segment type.

Section	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W	Patch
AcctGroup	R	O	N	N	N	N	N	N
COEServices	O	O	O	O	O	O	O	O
Community	O	O	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O	O	O
Compat	O	O	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O	O	O
Data	N	N	N	N	R	N	N	N
Database	X	X	X	X	X	X	X	X
Direct	O	O	O	O	O	O	O	O
FilesList	O	O	O	R	O	O	O	O
Hardware	R	R	R	R	R	R	R	R
Icons	R	O	N	O	N	N	O	O
Menus	R	O	N	O	N	N	O	O
ModName	*	*	*	*	*	*	*	*
ModVerify	*	*	*	*	*	*	*	*
Network	N	N	O	N	N	N	N	N
Permissions	O	O	N	N	N	N	O	O
Processes	O	O	O	O	N	N	O	O
ReqrdScripts	R	O	O	N	N	N	O	N
Requires	O	O	O	O	O	O	O	O
Security	R	R	R	R	R	R	R	R
SegType	*	*	*	*	*	*	*	*
R - Required	O - Optional		N - Not Applicable		X - Reserved for Future		* - Obsolete	

Table 2. SegInfo Descriptor Sections

4.3.3 Installing a Segment

Follow the procedures below to install a segment after it has been created.

Run VerifySeg

The VerifySeg tool must be run during the development phase to ensure segments use segment descriptor files properly. Run the VerifySeg tool whenever a segment is created or modified. When VerifySeg is run to verify a segment, a `Validated` file is created. This file is required to create the installation media or to use the TestInstall tool on the segment. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using VerifySeg.

Run TestInstall

Executing the TestInstall tool is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation on the developer's workstation before actual installation. The workstation must have the DII COE kernel installed before running TestInstall. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using TestInstall.

Run MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment(s) for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using MakeInstall.

Run COEInstaller

The COEInstaller tool installs a segment from tape, disk, or other electronic media. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using the COEInstaller.

4.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, although the segments may need to add menu items and icons. Some segments may need to add special extensions such as sockets. This subsection describes how to add menu items, icons, and special extensions.

4.4.1 Adding Menu Items

Menu Entry Format

The Menu Descriptor in the `SegInfo` file is used to specify the name of the segment's menu file and the name of the affected segment's menu file.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the entries in the named menu file. The format of the entries is in ASCII with colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. You may use any or all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs:

A **`PDMENU`** line contains the following elements:

```
PDMENU: name : enable flag : id # :
```

`PDMENU`	Keyword that indicates the start of a pull-down menu.
`name`	Text used to name the menu. The menu name is displayed on the menu bar.
`enable flag`	Integer value that indicates whether a menu is enabled or disabled. The enable flag is `1` if a menu is enabled or `0` if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.
`id#`	Optional integer value that provides a unique ID number for the menu. The `PDMENU id#` value must be unique within the menu description file. An absolute value may be provided. However, the `id#` field should be left empty so that relative numbering is used by default. With relative numbering, an `id#` of `R1` (or leaving the field blank) sets the menu's ID number to `1` plus the `id#` of the last menu processed. An `id#` of `R2` sets the menu's ID number to `2` plus the `id#` of the last menu processed.

The following is an example of a `PDMENU` line:

```
PDMENU: Map Options : 1 : R1 :
```

A **PDMENUEND** line contains the following element:

```
PDMENUEND:
```

PDMENUEND Optional keyword that indicates the end of a group of pull-down menu items. If PDMENUEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of a PDMENUEND line:

```
PDMENUEND:
```

An **ITEM** line contains the following elements:

```
ITEM: name : command : execution type : enable flag : # instances : id# :
check value : security char : autolog flag : print flag : disk flag :
```

ITEM — Keyword that indicates a menu item description line.

name — Text used to name the menu item. The item name is displayed in the pull-down menu.

command — Program with space-separated arguments that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.

execution type — Integer value that indicates how to execute a command, as follows:

- 1 = executable program
- 2 = void callback function with no parameters (not yet implemented)
- 3 = Motif callback function (not yet implemented).

enable flag — Integer value that indicates if a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.

 # instances — Integer value that is used to set the maximum number of times the item can be executed simultaneously.

id# — Optional integer value that provides a unique ID number for the menu item. Each ITEM id# entry must be unique within a PDMENU listing. (ITEM entries in a PRMENU must be unique within that PRMENU.) Refer to the id# description under the PDMENU keyword listing.

check value Optional integer value that sets the star and check annotations of a menu item. The possible values are:

- `0` = no annotation (default)
- `1` = visible check mark
- `2` = check mark, but not visible
- `3` = visible star
- `4` = star member, but not visible.

This element is not yet fully implemented.

security char Optional character value that is used to determine the lowest security level under which a menu item can be classified. Valid settings are:

- `N` = No Classification
- `U` = Unclassified (default)
- `C` = Confidential
- `S` = Secret
- `T` = Top Secret.

autolog flag Optional character value, `T` or `F`, used to indicate if the command should be logged automatically. This element is not yet fully implemented.

print flag Optional character value, `T` or `F`, used to indicate if the command should have a print capability. This element is not yet fully implemented.

disk flag Optional character value, `T` or `F`, used to indicate if the command should have disk access capability. This element is not yet fully implemented.

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **`PRMENU`** line contains the following elements:

```
PRMENU: name : enable flag : id# :
```

`PRMENU` Keyword that indicates a cascading menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.

name Text used to name the cascade menu with which to connect. The `PRMENU` name is displayed in the pull-down menu.

enable flag Integer value that indicates if a cascade menu is enabled or disabled. The enable flag is `1` if a cascade menu is enabled or `0` if it is disabled. A disabled cascade menu means that menu options on the cascade menu

cannot be selected.

id# Optional integer value that provides a unique ID number for the cascading menu. Each PRMENU id# must be unique within a PDMENU listing. Refer to the id# entry under the PDMENU keyword listing.

The following is an example of a PRMENU line:

```
PRMENU: Software : 1 : R1 :
```

A **CASCADE** line contains the following element:

```
CASCADE: name :
```

CASCADE	Keyword that indicates the start of a cascade menu. The cascade menu connects to the PRMENU entry of the same name.
name	Text used to name a cascade menu. The name is used to attach a cascade menu to a cascading button. This name must be the same as the name field in the PRMENU entry.

The following is an example of a CASCADE line:

```
CASCADE: Software :
```

A **CASCADEEND** line contains the following element:

```
CASCADEEND:
```

CASCADEEND	Optional keyword that indicates the end of a group of cascade menu items. If CASCADEEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of a CASCADEEND line:

```
CASCADEEND:
```

An **APPEND** line contains the following elements:

```
APPEND: name :
```

APPEND	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not exist already. The group is appended to the PDMENU or CASCADE entry of the same name.
name	Text used to select the menu to which a group of items is appended.

The following is an example of an APPEND line:

```
APPEND: Options :
```

An **APPENDEND** line contains the following element:

```
APPENDEND:
```

APPENDEND	Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If APPENDEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of an APPENDEND line:

```
APPENDEND:
```

A **SEPARATOR** line contains the following element:

```
SEPARATOR:
```

SEPARATOR	Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a SEPARATOR line:

```
SEPARATOR:
```

Example of Adding a Menu Item

To add menu items, include the Menus Descriptor in the SegInfo Segment Descriptor file. Specify the Menu file you use wish to load and the Menu file you wish to update. The Menu file you wish to load should be located in Menus directory of the segment. If your segment name is TstSeg, the file would be located in the TstSeg/data/Menus directory. The following example will add the Test Program menu item to the Software menu under the SysAdm account group by updating the SA_Default.main menu file.

The following file changes must be made to ensure the TSTCOEAskUser_example program is executed from the Software menu, Test Program option:

TstSeg/SegDescrip/SegInfo entry:

```
[Menus]
TstSegMenu:SA_Default.main
```

TstSeg/data/Menus/TstSegMenu entry:

```
#--------------------
# Software Menu Items
#--------------------
APPEND          :Software
ITEM            :Test Program   :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

The `$SEGMENT` keyword must be used in the `SegName` Segment Descriptor file to specify the name of the affected segment. In this case it is `System Administration`.

```
#
# SegName For the TstSeg segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

4.4.2 Adding Icons

Icon Entry Format

The Icon Description Entry contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the CDE. The entry must be available to the CDE at startup as part of the base set of icons.

Icons are built using the icon section in the `SegInfo` file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A # symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```

The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Window Title : Icon Path : Executable Name : Comments
```

(where `Window Title` is the title placed in the application window, `Icon Path` is the full path to the pixmap/xpm image, `Executable Name` is the name of the executable to be launched by the menu program, and `Comments` is an optional comment line)

An example of an affected icon file is as follows:

```
Edit Profiles : /h/AcctGrps/SysAdm/data/Icons/Prof.img:EditProfiles :
This is the EditProfiles icon
```

Example of Adding an Icon

To add an icon, include the `Icons` Descriptor in the `SegInfo` Segment Descriptor file. Specify the icon file you wish to load and the icon file you wish to update. The icon file you wish to load should be located under the `TstSeg/data/Icons` directory, assuming the segment's directory name is `TstSeg`. This example will add the `Test Program` icon to the SysAdm account group. When invoked through the icon, the program `TSTCOEAskUser_example` will be executed.

`TstSeg/SegDescrip/SegInfo entry:`

```
[Icons]
TstSegIcons:SA_Default
```

`TstSeg/data/Icons/TstSegIcon file:`

```
TstSegIcons
#--------------------
# Software Icons
#--------------------
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

Also include the `$SEGMENT` keyword in the `SegName` Segment Descriptor file to specify the name of the affected segment. In this case it is `System Administration`.

```
SegName

#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

4.4.3 Reserving a Socket

To add a service, include the `COEServices` Descriptor in the `SegInfo` Segment Descriptor file. Also include the `$SERVICES` keyword in the `SegInfo` Segment Descriptor file to specify the service to be added. If the port number requested is already in use under another name, an error will be generated.

NOTE: Port numbers in the range 2000-2999 are reserved for DII COE segments.

```
[COEServices]
#
#    This is my service to add
#
$SERVICES
irc_ser:3001:upd
```

4.4.4 Displaying a Message

This subsection shows an example of how to display a message during the PostInstall process. Five runtime tools can be used to communicate with a user: COEAskUser, COEInstError, COEMsg, COEPrompt, and COEPromptPasswd. These tools may be used to display information to the user or to ask the user a question and, based on the result, perform different actions.

In this example, the user is asked questions using the COEAskUser runtime tool, which is described in Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*.

```
        #!/bin/csh
        #============================================================
        # PostInstall                                  Tst 1.0 1/95
        #
        # Routine to perform necessary actions after TstSeg has been
        # loaded.
        #============================================================
        COEAskUser -B "RED LAN" "BLUE LAN" "Which LAN Will You Be Connecting
To"

        if ($status == 1) then
          COEAskUser -YN "On The RED LAN Do You Want Port #66?
        #
        # Perform Some Action Based On Results
        #
          exit(0)
        else if ($status == 0) then
          COEAskUser -YN "On The BLUE LAN Do You Want Port #6?
        #
        # Perform Some Action Based On Results
        #
          exit(0)
        else
          COEMsg "Invalid Return Status"
          exit(-1)
        endif
        exit(0)
```

Appendix A - Sample Segments

A hard copy of a software segment, called TstSeg, has been included as a basic example of a segment. As shown in Appendix B, *Verifying Segment Syntax and Loading a Segment onto Tape*, the sample segment will pass the checks performed by the COE tool VerifySeg. The sample segment will add the `Test Program` menu item to the SysAdm account group. When invoked through the menu item, the program `TSTCOEAskUser_example` will be executed.

Refer to Appendix B for instructions on how to validate the `TstSeg` segment and load the segment onto tape.

The layout of the sample segment is:

```
TstSeg:
./   ../   Scripts/   SegDescrip/  bin/  data/

TstSeg/Scripts:
./   ../   .cshrc.TST

TstSeg/SegDescrip:
./   ../   DEINSTALL   ReleaseNotes   SegInfo  SegName   VERSION

TstSeg/bin:
./   ../   TSTCOEAskUser_example

TstSeg/data:
./   ../   Menus/

TstSeg/data/Menus:
./   ../   TstSegMenu
```

NOTE: After the segment has passed VerifySeg, a validated file will be added to the `SegDescrip` directory.

The `Scripts` directory contains the following:

`.cshrc.TST`

```
#===========================================================================
# Define required runtime environment variables
#===========================================================================
setenv TST_HOME     /h/TstSeg
#
# Add bin to path
#
set path=($path $TST_HOME/bin)
```

The `SegDescrip` directory contains the following:

`DEINSTALL`

```
# !/bin/csh
#
# Deinstall For TstSeg
#
```

NOTE: The existence of the `DEINSTALL`, even if it does not contain any instructions, allows the segment to be deinstalled.

ReleaseNotes

```
#
# Release Notes For TstSeg
#
This is my Test Segment For Example Purposes
```

SegInfo

```
#
# SegInfo File For TstSeg
#

[Hardware]
$CPU:IBM
$MEMORY:200
$DISK:131

[Menus]
TstSegMenu:SA_Default.main

[Reqrd Scripts]
.cshrc:.cshrc.TST

[Security]
UNCLASS
```

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

TstSegMenu

```
#-------------------------
# Software Menu Items
#-------------------------
APPEND  :Software
ITEM  :Test Program  :TSTCOEAskUser_example:1:1:1:R1
APPENDEND:
```

VERSION

```
#
# Version Number For TstSeg
#
1.0.0.1 : 05/31/96: 10:08
```

Appendix B - Verifying Segment Syntax and Loading a Segment onto Tape

This appendix provides examples of how to convert a segment from the Joint Maritime Command Information System (JMCIS) format to the *DII COE Integration and Runtime Specification* segment format, verify segment syntax, install a segment temporarily, and load a segment onto an installation tape. The segment verification and loading process involves the following steps:

STEP 1: **Run the VerifySeg tool**. Run VerifySeg to validate that the segment conforms to the rules for defining a segment (i.e., to verify the segment syntax).

STEP 2: **Run the TestInstall tool**. Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 5.

STEP 3: **Run the MakeInstall tool**. Run MakeInstall to load the segment onto an installation tape. After the segment is loaded onto tape, it is ready to be installed using the Segment Installer option from the System Administration menu bar.

Subsections B.1-B.3 show how to perform these steps against the TstSeg sample software segment, which is described in Appendix A, *Sample Segments*.

NOTE: In the subsections below, the VerifySeg, TestInstall, and MakeInstall tools are being run against the `TstSeg` sample segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

`(F)`	indicates a FATAL ERROR	`(D)`	indicates a DEBUG statement
`(W)`	indicates a WARNING	`(V)`	indicates a VERBOSE statement
`(E)`	indicates an ERROR	`(O)`	indicates an ECHO statement

NOTE: In the following subsections, boldface text indicates information that the user must input.

B.1 Running VerifySeg Against the Sample Segment

```
*******************************************************************************

VerifySeg -p /home2/TestSegs TstSeg

Results of verification (/home2/TestSegs/TstSeg) :Totals
------

Errors:     0
Warnings:   0
*******************************************************************************
```

B.2 Running TestInstall Against the Sample Segment

```
*******************************************************************************

TestInstall -p /home2/TestSegs TstSeg

*******************************************************************************

TestInstall - Version 1.0.0.4

*******************************************************************************

The following options have been selected:

*******************************************************************************

Print warning messages.

*******************************************************************************

Segments to be TestInstalled:

*******************************************************************************

Segment: TstSeg Path: /home2/TestSegs

*************WARNING*************

TestInstall may modify COE files already in use. This may cause unpredictable
results if COE processes are already running. Make sure no other COE processes
are running before using TestInstall.

Do you want to continue with the TestInstall? (y/n):  y
Processing TstSeg
The segment /home2/TestSegs/TstSeg already uses the DII COE standard.
ConvertSeg is not required.

Successfully ran preprocessor on segment TstSeg

Do you want to run PreInstall for Segment TstSeg? (y/n):  y
Calling PreInstall Script
effective 0 real 0

Do you want to run PostInstall for Segment TstSeg? (y/n):  y
effective 0 real 0
Successful Installation of TstSeg
```

****

B.3 Running MakeInstall Against the Sample Segment

```
*******************************************************************************
```

MakeInstall -p /home2/TestSegs TstSeg

```
1  Write to disk
2  /dev/rmt/3mn            3 MBytes (HP DAT DT-30)
3  /dev/rmt/3mn          680 MBytes (HP DAT DT-30)
4  /dev/rmt/3mn         1360 MBytes (HP DAT DT-60)
5  /dev/rmt/3mn         2048 MBytes (HP DAT DT-90)
6  /dev/rmt/3mn         2730 MBytes (HP DAT DT-120)
7  /dev/rmt/3mn         4096 MBytes (HP DAT DT-210)
8  /dev/rmt/stn           60 MBytes (HP 1/4 inch DC600 Cartridge)
9  /dev/rmt/stn          150 MBytes (HP 1/4 inch DC6150 Cartridge)
10 /dev/rmt/stn          250 MBytes (HP 1/4 inch DC6250 Cartridge)
11 /dev/rmt/stn          525 MBytes (HP 1/4 inch DC6525 Cartridge)
12 /dev/rmt/1mn         1225 MBytes (HP 54M Exabyte)
13 /dev/rmt/1mn         2560 MBytes (HP 112M Exabyte)
14 /dev/nrst0            680 MBytes (Sun DAT DT-30)
15 /dev/nrst0           1360 MBytes (Sun DAT DT-60)
16 /dev/nrst0           2048 MBytes (Sun DAT DT-90)
17 /dev/nrst0           2730 MBytes (Sun DAT DT-120)
18 /dev/nrst0           4096 MBytes (Sun DAT DT-210)
19 /dev/nrst0             60 MBytes (Sun 1/4 inch DC600 Cartridge)
20 /dev/nrst0            150 MBytes (Sun 1/4 inch DC6150 Cartridge)
21 /dev/nrst0            250 MBytes (Sun 1/4 inch DC6250 Cartridge)
22 /dev/nrst0            525 MBytes (Sun 1/4 inch DC6525 Cartridge)
23 /dev/nrst0           1225 MBytes (Sun 54M Exabyte)
24 /dev/nrst0           2560 MBytes (Sun 112M Exabyte)
25 /dev/nrtape           680 Mbytes (SGI DAT DT-30)
26 /dev/nrtape          1360 Mbytes (SGI DAT DT-60)
27 /dev/nrtape          2048 Mbytes (SGI DAT DT-90)
28 /dev/nrtape          2730 Mbytes (SGI DAT DT-120)
29 /dev/nrtape          4096 Mbytes (SGI DAT DT-210)
30 /dev/rmt0.1          1225 Mbytes (Sun 54M Exabyte)
31 /dev/rmt0.1          2560 Mbytes (Sun 112M Exabyte)
32 /dev/nrmt0h           680 Mbytes (Sun DAT DT-30)
33 /dev/nrmt0h          1360 Mbytes (Sun DAT DT-60)
34 /dev/nrmt0h          2048 Mbytes (Sun DAT DT-90)
35 /dev/nrmt0h          2730 Mbytes (Sun DAT DT-120)
36 /dev/nrmt0h          4096 Mbytes (Sun DAT DT-210)
37 Other

Enter device to use (1, 2, etc) or type 'q' to quit.  1
Enter name of the output file or type 'q' to quit.  TstSeg

Processing Segment: /home2/TestSegs/TstSeg ...

Enter your name for the Tape Header:  John Smith
Enter a serial number for the Tape Header:  1
Enter any desired comment to put in the Tape Header (up to 255 characters) :
Test Load of TstSeg
```

```
Tape Index Attr Type Hardware Class Directory (Segment Name  - Version)
================================================================================
1      1    O    S      HP      U   /home2/TestSegs/TstSeg
                                     Test Segment  - 1.0.0.1
================================================================================
Attr : PS - Parent Segment  CS - Child Segment  0 - Other
Type : A - Acct Group       S - Software        C - COTS
       D - Data             P - Patch
Class: U - Unclassified     C - Confidential    S - Secret  T - Top Secret

Number of segments to write to tape: 1
Space required: 0.114 MByte (including Tape Header and Table of Contents)

********************************************************************************


********* Insert tape #1 *********

Press any key to continue.
0+1 records in.
1+0 records out.

************ DII Install tape completed ************
```

Appendix C - Installing the Developer's Toolkit

The Developer's Toolkit contains the components needed to create segments that use DII COE components. The Developer's Toolkit is distributed on magnetic media in relative tar format.

Developer's tools are delivered separate from the kernel tape. By default, these tools are located underneath the `/h/DII_DEV/bin` directory and are distributed as part of the Developer's Toolkit. Developer's tools can be run from the command line, and some can be run from other code using published APIs.

As distributed, the toolkit contains the following:

- API libraries
- C header files for public APIs
- On-line UNIX manual pages for some APIs
- COE development tools
- Sample segments
- Environment setup script.

The toolkit does not contain any products that require a license (e.g., compilers, editors, relational database management systems). The developer is responsible for acquiring these items as needed.

Follow the steps below to install the Developer's Toolkit.

STEP 1: **Log in**. Log in as the root user and enter the appropriate password.

STEP 2: **Change directories and install the Developer's Toolkit**. Type the following commands:

```
cd /h (or directory of choice)[RETURN]

tar xvfp [tape device] [RETURN]
```

NOTE: You must update the environment variable `LD_LIBRARY_PATH` to include Motif libraries in order to run graphical tools. For example:

```
set LD_LIBRARY_PATH=( $LD_LIBRARY_PATH /usr/lib/Motif1.2 )
```

Developers may install the toolkit on the disk in any directory they desire. After the tar is performed, all of the components of the Developer's Toolkit will reside under the `DII_DEV` directory. These components are listed below:

data files	`DII_DEV/data`
public header files	`DII_DEV/include`
public libraries	`DII_DEV/libs`
executables	`DII_DEV/bin`
manual pages	`DII_DEV/man`
scripts	`DII_DEV/Scripts`
examples	`DII_DEV/examples`
sample segments	`DII_DEV/SampleSegments`

Developers should include `DII_DEV/bin` in the path environment variable for their development environment. The `DII_DEV/man` directory should also be included in the search path for UNIX manual pages. Developers must source the `MakeTOOLSEnv` environment setup script. This will set up the following four environment variables: `MACHINE`, `MACHINE_CPU`, `MACHINE_OS`, and `TOOLS_HOME`. Read the `README` file at the top level of the `DII_DEV` directory for more information about these environment variables.

Developers are encouraged to submit tools to the COE community for inclusion in the Developer's Toolkit. All tools submitted must be license and royalty free and must include a manual page for on-line documentation. Developers who want to release source code for their contributed tools may do so, and the source code for each tool will be organized under the `DII_DEV/src` directory.

Reference Section 6.0, *Development Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of the development environment.

Appendix D - Installing Optional Common Desktop Environment Products

The Common Desktop Environment (CDE) provides windows, workspaces, controls, menus, and a front panel to help users organize and manage work. Follow the steps below to install optional CDE products:

STEP 1: **Log in**. Log in as `root`.

STEP 2: **Insert the CD into the drive**. Insert the TriTeal Enterprise Desktop (TED) CD into the CD drive.

STEP 3: **Mount the CD.** Mount the CD onto `/cdrom`.

STEP 4: **Change to the `cdrom` directory**. Type the following command to change to the `cdrom` directory:

```
cd /cdrom [RETURN]
```

STEP 5: **Ensure that the Xwindow `DISPLAY` environment variable is set**. Type the following command to ensure that the Xwindow `DISPLAY` environment variable is set:

```
setenv DISPLAY unix:0.0 [RETURN]
```

STEP 6: **Ensure that Xwindow clients can connect to the display**. Type the following command to ensure that Xwindow clients can connect to the display:

```
xhost + [your machine name]
```

STEP 7: **Select the items to install**. Type the following command:

```
./install [RETURN]
```

The `Package Selections` window appears.

The following check boxes appear in the `Package Selections` window. Select the items you want to install.

- C TED Runtime
- C TED Development
- C TED Fax
- C TED Locales

- TED Vision
- WinTED.

Next to each of the checkboxes is a push button that can be used to customize the installation for that particular item.

For example, when installing the TED runtime item, you can configure it to install any of the following four items:

- TriTeal Enterprise Desktop
- TED Postscript manuals
- TED help
- TED manual pages.

NOTE: The TriTeal Enterprise Desktop MUST NOT be installed. This item, instead, is installed with the kernel. If you reinstall this item, you will remove the CDE segment.

STEP 8: **Install the selected items**. Select the `YES - Edit System Files` option after selecting all the items to install. Execute `Install` from the main screen.

STEP 9: **Execute post install**. Select the `Execute Post Install` option from the `File` pull-down menu to execute post install.

STEP 10: **Exit the menu**. Select the `Exit` option from the `File` menu.

STEP 11: **Remove the CD.** Type the following commands to unmount and remove the CD:

```
cd / [RETURN]

umount /cdrom [RETURN]
```